Е.В. Касьянова*

ВВОДНЫЙ КУРС ПРОГРАММИРОВАНИЯ НА БАЗЕ ЯЗЫКА ZONNON

ВВЕДЕНИЕ

При обучении программированию наиболее важным представляется начальный этап, на котором обучаемый должен овладеть навыками точного формулирования алгоритмов на языках высокого уровня. Это невозможно сделать, прочитав несколько руководств или прослушав курс лекций по программированию. Необходима практика конструирования алгоритмов, и здесь невозможно обойтись без языка программирования, пригодного для целей начального обучения, и подходящего набора примеров и задач.

В статье кратко представлен вводный курс программирования на базе языка Zonnon, работа над которым ведется в Цюриховском институте информатики [6, 11, 12]; описаны две книги: «Введение в программирование» и «Практикум по программированию» [2, 3], поддерживающие этот курс.

Zonnon — это новый универсальный язык программирования в семействе языков Паскаль, Модула-2 и Оберон. Он сохраняет стремление к простоте, ясному синтаксису и независимости концепций, а также уделяет внимание параллельности и легкости композиции и выражения. Унификация абстракций является стержнем проектирования языка Zonnon, и она отражается в его концептуальной модели, основанной на модулях, объектах, определениях и реализациях. Язык Zonnon содержит такие новые черты, как активность в объектах, основанный на межобъектном взаимодействии диалог, перегрузка операций и обработка исключительных ситуаций. Язык Zonnon специально разрабатывается как платформно-независимый язык. Первая реализация языка Zonnon выполнена для платформы .NET [15]. Кроме того, предполагается интеграция компилятора в систему программирования Visual Studio .NET (в сотрудничестве с компанией Microsoft) [1].

Курс опирается на опыт преподавания основного курса по программированию для студентов механико-математического факультета (ММФ) Но-

* kev@iis.nsk.su

Работа выполнена при финансовой поддержке Microsoft Research (грант «Разработка вводного курса программирования»).

восибирского государственного университета (НГУ) [4]. Это семестровый курс, который предполагает еженедельно шесть часов аудиторных занятий:

- два часа лекционных занятий,
- два часа семинарских занятий у доски и
- два часа практических занятий за компьютером.

В основу курса положен принцип концентрического изложения материала. Предполагается, что с первых занятий студенты начинают упражняться в составлении программ, которые могут реально выполняться на доступных ЭВМ, и постепенно овладевают навыками разработки на языке Zonnon линейных, ветвящихся и итеративных алгоритмов, алгоритмов с массивами и процедурами. Также постепенно, одновременно с расширением класса решаемых задач, студенты углубляют свои знания о языке Zonnon.

К концу вводной части курса студенты овладевают основными конструкциями языка, образующими то естественно выделяемое его ядро для «начального обучения», которое условно можно назвать языком мини-Zonnon. В этом ядре нет ряда весьма важных возможностей языка Zonnon, таких как, например, поддержка объектно-ориентированного программирования, сужены средства языка Zonnon по структурированию данных и действий — например, нет множеств и бесконечных циклов. Поэтому, хотя мини-Zonnon и содержит все необходимые сведения о языке Zonnon для построения программ начального обучения, он является лишь базой для изучения программирования на языке Zonnon. Отметим, что в мини-Zonnon идентификаторы могут содержать наряду с латинскими буквами и буквы русского алфавита, хотя это и запрещено в языке Zonnon.

Следует отметить, что использование в качестве основы вводной части курса подмножества «живого» языка программирования на ММФ НГУ является традиционным. Первым использовалось подмножество языка Алгол-60, получившее название языка Минал [5]. Затем, когда в НГУ появились терминальные классы с диалоговым языком, являющимся вариантом языка Фортран, Минал был заменен языком мини-Фортран [8]. В начале 80-х гг. прошлого столетия в НГУ появилась одна из первых реализация языка Паскаль, тогда еще на машинах серии ЕС [7], и с тех пор вплоть до настоящего времени место первого учебного языка программирования для студентов ММФ НГУ бессменно занимает язык мини-Паскаль [4].

Статья начинается с изложения основных особенностей языка Zonnon (разд. 1). Разд. 2 содержит описание цели и основных принципов курса. Разд. 3 является кратким описанием книги «Введение в программирование», являющейся учебником по курсу. Цели и содержание компьютерного

практикума составляют разд. 4. Разд. 5 содержит описание книги «Практикум по программированию», предназначенной для поддержки практикума. В заключении подводятся некоторые итоги разработанного курса. Полный синтаксис языка Zonnon в состоянии на октябрь 2004 г. приведен в приложении.

1. ЯЗЫК ПРОГРАММИРОВАНИЯ ZONNON

Сам проект по разработке языка Zonnon возник как продолжение работы его авторов над языком Оберон (Oberon) в проекте, который был начат Microsoft Research в 1999 году с целью реализации значительного числа нестандартных языков программирования для платформы .NET [9, 10]. Язык Оберон [13, 14] является хорошо известным преемником языков Паскаль (Pascal) и Модула-2 (Modula-2). Мотивация авторов на продолжение работы по развитию языка Оберон связана со следующими двумя целями [11]:

a) экспериментально исследовать потенциальные возможности платформы .NET в комбинации с новой технологией CCI интеграции компиляторов для проектирования языков;
b) реализовать для платформы .NET язык Zonnon, являющийся эволюцией языка Oberon для .NET.

Поскольку язык Zonnon задуман как дальнейшая эволюция языка Оберон, авторы стремятся сохранить такие важные черты языка Оберон и его преемников, как компактность языка, ясность, недвусмысленность и ортогональность его основных понятий. Вместе с тем, чтобы создать современную альтернативу языку Оберон, авторы внесли в язык ряд изменений, основными из которых являются:

- более развитая модульная структура языка;
- продвинутая и одновременно простая и ясная объектная модель;
- концепция активных объектов.

«Общеалгоритмическая» часть языка Zonnon практически полностью повторяет соответствующие части языка Оберон, поэтому данный язык можно рассматривать как естественную замену Оберону там, где последний традиционно используется для обучения программированию.

Язык Zonnon возник как естественный результат исследований, проводившихся в течение последних нескольких лет в Федеральном Технологическом Институте (ETH) в Цюрихе. Непосредственными предшественни-

ками языка следует считать Active Oberon, реализованный как базовый язык ядра операционной системы BlueBottle, а также реализацию языка Oberon для платформы .NET (Oberon.NET). Язык Zonnon, имея ряд общих черт с указанными языками, вместе с тем существенно отличается от них по ряду принципиальных моментов. Первая реализация Zonnon выполнена для платформы .NET. Кроме того, предполагается интеграция компилятора в систему программирования Visual Studio .NET (в сотрудничестве с компанией Microsoft).

Хотя по размеру Zonnon уступает таким языкам, как C#, Java и Ada (см. приложение), он является универсальным языком программирования, пригодным для широкой области приложений. Типично она включает компонентно-ориентированную композицию, параллельные системы, алгоритмы и структуры данных, объектно-ориентированное и структурное программирование, графику, математическое программирование и низкоуровневое системное программирование. Zonnon предоставляет богатую объектную модель с инкапсулированным поведением и синтаксически-управляемыми диалогами, которые инкапсулируют состояние. Он может использоваться для написания программ в традиционном и объектно-ориентированном стилях, а также весьма подходит для целей обучения программированию от его базовых принципов до продвинутых концепций.

Унификация абстракций является стержнем проектирования языка Zonnon. Она отражается в его четырех столпах:

- *модуль (module)* — текстовый контейнер, а также объект композиции программ;
- *объект(object)* — типовой образец для определяемых объектов;
- *определение (definition)* — концепция абстракции и композиции для определяемых интерфейсов;
- *реализация (implementation)*— контейнер для переиспользуемых фрагментов объектных реализаций.

Модуль (*module*) имеет двойственную природу: он объявляет синтаксический контейнер для логически связанных программных деклараций и одновременно определяет объект, чей жизненный цикл управляется системой. Таким образом, модель предоставляет механизм для текстуального разбиения исходной программы, а также динамической загрузки на этапе выполнения некоторой части программы в виде созданного экземпляра объекта.

Любое число динамически созданных объектов может иметь свои жизненные циклы, управляемые системой, однако в любое заданное время

только единственный экземпляр каждого модульного объекта может быть создан системой. Поскольку модуль формирует единицу инкапсуляции и скрытия данных, он представляет собой также идеал как контейнер для реализации абстрактных типов данных.

Объектом (*object*) является типовой образец, включающий в себя поля, методы и активности. Поля представляют состояние объекта, методы — его функциональность, а активности — его параллельное поведение. Он может выставлять свой интерфейс к системному окружению двумя способами. Во-первых, с помощью присущего ему *интерфейса* (*intrinsic interface*), т.е. множества всех элементов, которые программист выбрал сделать публичными, а не сохранять как приватные, и, во-вторых, большим числом *определений* (*definitions*), каждое из которых выставляет свой *аспект* (*facet*), представляющий собой некоторый взгляд на службы объекта со стороны клиентов.

Определение (*definition*) задает некоторый свой *аспект* (*facet*) объекта в терминах абстрактного интерфейса, включающего определения полей и сигнатуры методов. Определения могут образовывать не только иерархию, но и *сеть связанных типов* (*a network of related types)*.

Реализация (*implementation*) определяет агрегат реализаций полей и методов, предназначенных для переиспользования, при присоединении к программе с помощью одного или нескольких объектных образцов. Объект, реализующий определение, должен реализовывать все его поля и методы. Однако если объект импортирует реализацию определения с тем же именем, то это неявно предполагает, что она будет его (возможно частичной) реализацией.

Программный текст (*program text*) состоит из модулей, объектов, определений и реализаций. *Внутренний интерфейс* (*intrinsic interface*) программы представляет собой множество деклараций, сделанных публичными всеми ее частями. П*рограмма периода исполнения* (*runtime program*) состоит из одного или более модулей и любых объектов, которые создаются динамически. *Система* (*system*) предоставляет механизмы динамической программной загрузки и выгрузки модулей и динамическим управлением объектных ресурсов во время выполнения, когда происходит исполнение программы.

Эти конструкции используются для формирования всей структуры программы в виде программных *единиц: module, object*, *definition* и *implementation*. Каждая конструкция может существовать как *отдельно скомпилированная единица* (*separately compiled unit*) или может текстуально встраиваться в содержимое другой конструкции. Указанные единицы обеспечи-

вают базис для композиции программ при «программировании в большом», а также для текстуального разбиения и раздельной компиляции во время разработки программы.

Объектная модель в языке Zonnon основывается на концепции, что «все есть объект». Она поддерживает три точки зрения на объекты: во-первых, как сущности с некоторым внутренним типом, использующие абстрактные операции способом, безопасным для типов, во-вторых, как провайдеры служб, доступные через определенные интерфейсы, и, в-третьих, как автоматические агенты, взаимодействующие через формальные диалоги. *Активности (activities)* используются как для добавления поведения объектам, так и для реализации диалога. Они внедряют естественным образом параллелизм в язык.

Многие из концепций языка Zonnon получены им по наследству. Цель состояла в том, чтобы предложить выразительные и связующие черты, которые уже доказали свою ценность. Язык Zonnon также вводит некоторые новые черты, такие как перегрузка знаков операций для естественного представления математических и других выражений и обработка ситуаций для повышения надежности. Некоторые черты были им реанимированы (взяты из ранних членов семейства паскалеподобных языков), например, пары *definition*, *implementation* и типы перечисления языка Модулы-2 и, по прагматическим причинам, основная форма операторов *read* и *write* языка Паскаль.

Когда осуществляется выбор языка для построения современных программных систем, важным соображением является достижение взаимодействия между программами, написанными на разных языках. Язык Zonnon специально проектировался как платформно-независимый язык, поддерживающий такое взаимодействие.

2. ЦЕЛЬ И ОСНОВНЫЕ ПРИНЦИПЫ КУРСА

Курс предназначен для обучения основным методам построения корректных, эффективных и надежных программ на базе языка Zonnon и платформы Microsoft.NET. Он ориентирован на широкий круг лиц, обучающихся методам программирования, в первую очередь, на студентов НГУ, других вузов и средних учебных заведений, а также школьников, желающих углубить свои знания по программированию. Курс предназначен главным образом для тех учебных заведений, в которых

- в настоящее время используется язык Паскаль в качестве языка начального обучения программированию и
- есть желание перейти к более современному курсу программирования, охватывающему концепции языков программирования нового поколения, таких как Java и C#, но осуществить этот переход плавно, без резкого изменения сложившегося стиля преподавания программирования.

Разработанный курс базируется на ряде методических и технологических принципов, основными из которых являются следующие.

1. Принцип концентрического изложения материала, когда обучаемый осваивает языковые средства и приемы программирования постепенно, слой за слоем. При этом с первых занятий студенты начинают упражняться в составлении программ, которые могут реально выполняться на доступных им компьютерах, а освоение нового слоя означает просто расширение круга задач, которые может решать обучаемый.

2. Принцип обучения конструированию программ на подробно комментированных образцах решения тщательно подобранных задач. Назначение примеров — не только дать образцы и описать основные схемы алгоритмов, но и на сравнительном анализе разных решений одной и той же задачи познакомить студента с такими понятиями, как эффективность, наглядность и надежность решения.

3. Принцип доказательного программирования, когда программа строится вместе с доказательством ее правильности. Для этого в курсе вводятся понятия промежуточных утверждений и инвариантов, а в разрабатываемых алгоритмах решения задач такие утверждения записываются в форме программных комментариев.

4. Принцип пошаговой разработки программ, когда программа строится из формальной спецификации задачи с помощью мелких формально проверяемых шагов преобразования.

5. Принцип модульного программирования, позволяющий проектировать, разрабатывать и собирать программу по частям и с использованием библиотек уже готовых частей.

6. Принцип объектно-ориентированного программирования, позволяющий разработчикам программ легко создавать все более сложные приложения с помощью инкапсуляции, наследования и полиморфизма.

3. ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

Книга [2] предназначена для изучения студентами основных понятий программирования и постепенного овладения навыками разработки на языке Zonnon линейных, ветвящихся и итеративных алгоритмов, алгоритмов с процедурами и со структурированными данными.

Книга содержит порядка 3000 задач и состоит из 6 глав.

Внутри одной главы задачи сгруппированы по методам решения и языковым конструкциям, на освоение которых они ориентированы. Каждая группа составит самостоятельный параграф, содержащий помимо текстов задач либо описание соответствующих языковых понятий, либо примеры решения некоторых типичных задач данной группы. Назначение примеров — не только дать образцы и описать основные схемы алгоритмов, но и показать, как алгоритм может быть выведен из рекурсивных соотношений — спецификации алгоритма; привести инварианты циклов и другие промежуточные утверждения, из которых выводится правильность программы. Здесь же на сравнительном анализе разных решений одной и той же задачи студент знакомится с такими понятиями, как эффективность, наглядность и надежность решения.

Главу завершает список заданий. Каждая из формулировок заданий представляет собой фактически схему для построения конкретных вариантов задания: индивидуальных задач на программирование для всех студентов учебной группы. Эти варианты получаются в результате подстановки в текст вместо номера варианта его значения и выбора по значению номера варианта подходящих частей текста, составляющего формулировку задания.

Например, 7-й вариант следующего задания:

«в заданной последовательности целых чисел найти

номер (при N **mod** 3 = 0),
модуль (при N **mod** 3 = 1),
квадрат (при N **mod** 3 = 2)

такого

первого (при N **mod** 4 = 0),
последнего (при N **mod** 4 = 1),
минимального (при N **mod** 4 = 2),
максимального (при N **mod** 4 = 3)

элемента, который является

четным (при N **mod** 2 = 0),
не кратным N (при N **mod** 2 = 1)

числом и совпадает с кодом некоторой
буквы (при N **mod** 8 > 3),
цифры (при N **mod** 8 ≤ 3)»

будет иметь вид: «в заданной последовательности целых чисел найти модуль такого первого максимального элемента, который является не кратным 7 числом и совпадает с кодом некоторой буквы» .

Гл. 1 содержит вводные понятия и начинается с ответов на такие вопросы, связанные с алгоритмами и функциями, как: что такое алгоритм, что такое компьютер, как можно определять функции и как можно определить язык Zonnon?

Методам построения простейших программ посвящена гл. 2. В ней рассматриваются стандартные типы данных и средства организации вычислений в линейных программах. Излагаются вопросы доказательства свойств программ: поясняется, зачем нужны доказательства правильности программ, вводятся понятия промежуточных утверждений, внешней спецификации программы, полной и частичной правильности программ, описываются методы задания внешней спецификации простых программ, и излагается метод промежуточных утверждений доказательства их свойств. Приводятся образцы разработки линейных программ на примере решения таких задач, как периметр и площадь прямоугольного треугольника, симметричная буква, возведение в степень и площадь треугольника.

Гл. 3 посвящена методам построения простых программ без циклов. Она начинается с описания средств для организации и описания ветвлений в программах: вводится блок-схемные представления программ и их фрагментов, описываются условные операторы и операторы выбора, излагаются методы доказательства свойств ветвящихся программ. Затем приводятся образцы разработки ветвящихся программ на примере решения таких задач, как точка в треугольнике, максимум из трех чисел, табличное задание функции, анализ квадратного уравнения и определение типа треугольника.

Гл. 4 посвящена методам построения итеративных программ. В ней описываются средства для организации и анализа свойств циклических вычислений: операторы цикла с условием на продолжение, операторы цикла с условием на окончание и цикла с параметрами. Излагаются методы пошаговой разработки программ. Приводятся образцы решения задач независимой обработки элементов последовательности (перекодировщик, выборка элементов последовательности), вычисления элементов последовательности (нахождение факториала, вычисление числа e, схема Горнера), реализации функций на последовательностях (подсчет вхождений, количество мак-

симумов), обработки последовательности последовательностей и реализации двойных циклов (обработка слов предложения, приближенное вычисление сумм рядов).

Вопросам построения программ обработки структурированных данных посвящена гл. 5. В ней описываются средства задания и анализа программ со структурами данных, приводятся образцы решения задач обработки векторов (векторная функция, поиск в упорядоченном векторе, подсчет количеств вхождений букв, свертка вектора) и программы обработки матриц (поиск максимума в матрице, произведение матриц, преобразование матриц, поиск номера строки-серии).

Гл. 6 посвящена классу программ с процедурами и функциями. В ней рассматриваются правила описания и вызовов процедур, блоки и локализация имен, изменение действий (входные параметры) и получение результатов (выходные параметры) процедур, а также вычисление единственного значения (функции). Излагаются методы использования процедур и функций для пошаговой разработки программ, рассматриваются рекурсивные подпрограммы и исследуется, когда и как нужно использовать рекурсию, описывается метод структурной индукции для доказательства свойств рекурсивных программ. Приводятся образцы решения задач с использованием процедур и функций, таких как обработка последовательности векторов, Ханойские башни, кратные суммы, быстрая сортировка и числа Фибоначчи.

4. ЦЕЛИ И СОДЕРЖАНИЕ ПРАКТИКУМА

Основной задачей компьютерного практикума является не только и не столько обучение студентов собственно записи (кодированию) известного алгоритма на языке Zonnon для платформы .NET, а практическое закрепление знаний, получаемых в ходе лекций и семинарских занятий по вводному курсу программирования, и овладение общими методами, приемами и навыками технологии решения задач на компьютере. Основная цель, которая ставится перед студентом при выполнении индивидуальных заданий, составляющих компьютерный практикум, — это практическое освоение всех этапов разработки надежной и наглядной интерактивной (диалоговой) Zonnon-программы для компьютерного решения несложной задачи, требующей разработки алгоритма, обработки сложных структур данных и создания дружественного интерфейса.

В силу этого, тематика индивидуальных заданий для компьютерного практикума определяется, в первую очередь, всеми видами работ, которые должен освоить студент, чтобы научиться создавать качественные (эффективные, наглядные и надежные) нетривиальные программы. В большинстве случаев задача, решаемая во время выполнения индивидуального задания, — это задача невычислительного характера, имеющая краткую и точную (содержательную) формулировку и допускающая большое разнообразие решений.

- Вместе с тем студенту и преподавателю нужно иметь в виду, что многие из методов и понятий, используемых в индивидуальных заданиях, являются неотъемлемой частью образования современного специалиста, использующего компьютер для решения своих задач. Поэтому предполагается, что при прохождении компьютерного практикума студент получит не менее пяти индивидуальных заданий, по одному из каждого тематического раздела.

Выполнение каждого задания, составляющего компьютерный практикум, в общем случае включает следующие виды работ:

- анализ условия задачи и выработка подхода к ее решению;
- пошаговая разработка (на основании выбранного подхода) алгоритма решения и его описание;
- обоснование алгоритма;
- выбор и обоснование представления для входных, выходных и промежуточных данных;
- кодирование алгоритма, т.е. его запись на языке Zonnon;
- выбор и обоснование набора тестов, на которых будет проверяться программа;
- отладка программы и демонстрация ее правильной работы на выбранном наборе тестов.

Это разбиение условно в том смысле, что фактически некоторые виды работ тесно переплетаются и выполнение их обычно составляет единый процесс. Например, строить набор тестов удобнее одновременно с построением самого алгоритма, а обосновывать правильность работы алгоритма удобно путем детальной демонстрации процесса его построения.

Выполнение каждого задания студент завершает составлением отчета, который включает:

- формулировку задачи;

- описание программы для пользователя, ее внешнюю спецификацию, т.е. описание способа задания входных данных, вида результатов программы при заданных входных данных и сценария диалога в процессе исполнения программы;
- словесное описание алгоритма и обоснование его правильности и эффективности;
- текст программы;
- описание тестового набора и его обоснование.

Предполагается, что студент, решая задачу, создает интерактивную (диалоговую) Zonnon программу для платформы .NET, которая за один запуск может обрабатывать не один входной набор, а последовательность входных наборов произвольной длины. Эти входные наборы либо заранее размещаются в специальных "входных" файлах программы, предъявляемых преподавателю вместе с разработанной программой, либо задаются пользователем программы в процессе ее исполнения. Таким образом, разработанная студентом программа после завершения обработки очередного входного набора в зависимости от указания пользователя может либо завершить свою работу (остановиться), либо продолжить ее и перейти к обработке следующего входного набора. В последнем случае программа вступает в диалог с пользователем, в процессе которого пользователь программы готовит и инициирует новый счет по программе. Для этого он либо вводит очередной входной набор с помощью клавиатуры и мышки, либо дает указание программе, из какого файла ей нужно взять необходимые данные.

При этом совсем не требуется, чтобы при переходе программы к обработке следующего входного набора этот набор каждый раз целиком задавался заново. Например, во многих заданиях удобно разделить входной набор на две части, выделяя более общие (как правило, более объемные) исходные данные задачи в отдельную часть, и предусмотреть специальный вид реализации счета по программе на последовательности входных данных, различающихся второй частью, без необходимости повторного задания первой. Например, в задаче нахождения кратчайшего пути между двумя заданными вершинами заданной системы дорог более общей (и более объемной) частью входного набора является система дорог. Поэтому естественно решение, при котором система дорог является общей для нескольких следующих друг за другом счетов по программе. В этом случае каждый раз, переходя к новому счету по программе, пользователь может выбрать один из двух вариантов продолжения работы:

- осуществить ввод очередной системы дорог из заданного файла с выводом его изображения на экран,

- запустить нахождение кратчайшего пути между двумя заданными городами в текущей системе дорог.

Важно, чтобы до выполнения заданий студент и преподаватель согласовали все требования, предъявляемые к их выполнению, в том числе, виды работ по каждому заданию, а также порядок и сроки сдачи заданий. В любом случае понимание условий задач студентом не должно отличаться от понимания преподавателем. Поэтому раньше, чем студент начнет разрабатывать алгоритм, студент и преподаватель должны вместе тщательно проанализировать условие задачи и обсудить особенности и трудности ее решения, зафиксировать интерфейс программы, включая представление входных и выходных данных.

Необходим также постоянный контроль текущего состояния решения заданий для координации установленных преподавателем требований и действительных намерений студента. Необходимо уделять особое внимание высокой надежности создаваемых программ. Программа должна содержать достаточное число так называемых стопоров ошибок — динамических проверок справедливости утверждений, характеризующих правильность функционирования программы и ее применения. Очень важно, чтобы программа не только давала правильные результаты для корректных исходных данных, но и осмысленно реагировала на некорректные данные и указания пользователя. Не менее важно, чтобы программа выдавала на экран (или в специальный файл) в удобном виде информацию о ходе вычисления по программе в таком объеме (и в таком виде), который позволяет легко идентифицировать ошибку (как в программе, так и во входных данных) и локализовать место ее возникновения.

Программа обязательно должна быть наглядной, т.е. хорошо структурированной, с достаточным количеством комментариев и т.д., а также содержать инструкцию по своему использованию («встроенный help»).

Особое внимание нужно уделить правильности программы и полноте тестового набора. Описание каждого теста помимо файла с входными данными должно включать информацию, достаточную для оценки правильности работы программы на этих входных данных. Минимальное требование к тестовому набору состоит в том, что каждый оператор программы должен быть достижим (т.е. выполняться) хотя бы на одном тесте из этого набора. Поэтому среди тестов набора должны быть представлены и некорректные исходные данные.

Входные и выходные данные разработанной программы должны быть естественными для человека. Степень естественности представления определяется объемом той работы, которая требуется для перехода от содержа-

тельного описания входных данных к их представлению (очень часто неестественность входного представления проявляется в его неоправданно большом размере) и от представления выходных данных к их содержательному пониманию.

Задачи, составляющие индивидуальные задания, допускают, как правило, целый спектр различных по эффективности решений. Предполагается, что студент должен выбрать и обосновать по возможности хорошее решение. Минимальным требованием к эффективности решения является успешная работа программы на согласованном с преподавателем тестовом наборе.

Для удобства все индивидуальные задания разбиты на три группы:

- обычная сложность (их формулировки не имеют пометки),
- повышенная сложность (с пометкой "↑"),
- пониженная сложность (с пометкой "↓").

При оценке сложности задания рассматривались три показателя:

- сложность структур данных,
- сложность вычислений,
- изобретательность.

В показатель «изобретательность» включались такие свойства задания, как непривычность для студента понятий, используемых в задании, сложность извлечения из определений тех свойств, на которых должен базироваться алгоритм решения задания, а также сложная связь между структурами данных и вычислениями. Каждый из указанных трех показателей задания оценивался в баллах 0, 1 или 2. Пометку "↓" получили задания, набравшие в сумме по трем показателям всего 1 балл, а пометку "↑" — задания, суммарный балл которых больше 3.

4. ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ

Книга [3] предназначена для проведения компьютерного практикума по курсу программирования. Она содержит 500 индивидуальных заданий различной сложности, ориентированных на приобретение студентами навыка практического решения задач создания пользовательских и Web-приложений для платформы Microsoft.NET, требующих разработки алгоритма, обработки сложных структур данных и разработки дружественного интерфейса. Каждое индивидуальное задание — это самостоятельная, как правило, комбинаторная или логическая задача с краткой и четкой форму-

лировкой, не содержащей описания алгоритма. Тематические задачи разбиты на пять разделов по 100 заданий в каждом разделе:

- графы и системы дорог;
- грамматики, языки и автоматы;
- формулы и программы;
- геометрия;
- игры и модели.

Основную часть книги составляют подробные рекомендации по выполнению разных видов работ, которые должен освоить студент, чтобы научиться создавать эффективные, наглядные и надежные нетривиальные приложения для платформы Microsoft.NET, в том числе для работы в среде Интернет.

Книга состоит из введения и 6 глав.

Нужно сказать, что при составлении сборника авторы стремились сделать изложение доступным для читателей, не обладающих специальными знаниями ни по математике, ни по программированию. Поэтому все необходимые для решения задач сведения по программированию, в том числе и полное описание языка Zonnon (гл. 1), включены в сборник, а условия большинства заданий и упражнений доступны для учащихся старших классов. Приведенные в сборнике (гл. 2) определения понятий, встречающихся в заданиях, вполне достаточны, для того чтобы можно было понимать формулировки заданий (гл. 6). Однако для построения эффективного решения некоторых заданий может оказаться полезным знание некоторых свойств определяемых здесь математических объектов. Поэтому во введении не только описываются цели и задачи практикума и даются общие рекомендации преподавателю и студенту, но и дается обзор дополнительной литературы, которая может помочь студенту при выполнении его заданий.

Гл. 3, 4 и 5 содержат подробные рекомендации по решению индивидуальных заданий, составляющих практикум. В гл. 3 даются общие рекомендации по разработке и обоснованию алгоритма, представлению данных, анализу алгоритма и его сложности, выбору и обоснованию тестов, начиная с анализа условия задачи и кончая отладкой созданной программы. Особенностью заданий данного практикума является то, что они связаны со сложными структурами данных, с организацией поиска или порождения тех элементов конечного (но, как правило, чрезвычайно большого) множества, которые удовлетворяют определенным ограничениям и требуют при своем решении анализа алгоритма и его сложности. В гл. 4 приводятся рекомендации по алгоритмическому решению задач, связанных с организацией вычислений на дискретных конечных математических структурах. В ней

описываются основные типы данных, возникающие при выполнении заданий (списки, стеки, очереди, деревья, графы, орграфы), приводится язык описания алгоритмов, рассматривается поиск с возвращением, описываются алгоритмы поиска с возвращением, обходов ордерева в глубину и в ширину, обходов графа в глубину и в ширину, дается усовершенствование поиска с возвращением, приводятся методы ветвей и границ, динамического программирования, а также порождения объектов, перестановок, подмножеств множества, сочетаний. В гл. 5 рассматриваются вопросы выбора представления для структур данных, присутствующих в формулировках заданий, и даются конкретные рекомендации по выбору представлений для таких структур данных, как множество точек на плоскости, прямая на плоскости, окружность и многоугольник, последовательность, вектор, матрица, тексты, слова и предложения, граф, корневое дерево, система дорог, грамматика, автоматная диаграмма, логическая формула, логический фрагмент, простое выражение, полином, игры.

ЗАКЛЮЧЕНИЕ

В статье представлен вводный курс по программированию на базе языка Zonnon, работа над которым ведется в Цюриховском институте информатики. Курс опирается на опыт преподавания основного курса по программированию для студентов механико-математического факультета НГУ с использованием языка Паскаль. Язык Zonnon задуман как дальнейшая эволюция хорошо известного и широко применяемого на западе в учебных целях языка Оберон, являющегося преемником языков Паскаль и Модула-2. Развивая язык Оберон, исходя из современных потребностей в программировании, авторы сохранили в языке Zonnon такие важные черты Оберона и его предшественников, как компактность языка, ясность, недвусмысленность и ортогональность его основных понятий. Поэтому можно ожидать, что язык Zonnon и данный вводный курс будут востребованы теми учебными заведениями, которые в настоящее время используют Паскаль в качестве языка начального обучения программированию и имеют желание перейти к более современному курсу программирования, охватывающему концепции языков программирования нового поколения, таких как Java и C#, но осуществить этот переход плавно, без резкого изменения сложившегося стиля преподавания программирования.

СПИСОК ЛИТЕРАТУРЫ

1. **Джонсон Б., Скибо К., Янг М.** Основы Microsoft Visual Studio .NET 2003. — М.: Русская Редакция, 2003.
2. **Касьянов В. Н., Касьянова Е.В.** Введение в программирование. — `http://pco.iis.nsk.su/ICP`
3. **Касьянов В. Н., Касьянова Е.В.** Практикум по программированию. — `http://pco.iis.nsk.su/ICP`
4. **Касьянов В.Н.** Курс программирования на Паскале в заданиях и упражнениях. — Новосибирск: НГУ, 2001.
5. **Касьянов В.Н.** Язык программирования Минал. — Новосибирск: НГУ, 1979.
6. **Касьянова Е.В.** Язык программирования Zonnon для платформы .NET // Программные средства и математические основы информатики. — Новосибирск: ИСИ СО РАН, 2004. — С.189–205.
7. **Основы** языка ПАСКАЛЬ-360 / Сост. Касьянов В.Н. — Новосибирск: НГУ, 1982.
8. **Программирование** на мини-Фортране / Сост. Касьянов В.Н. — Новосибирск: НГУ, 1981.
9. **Просиз Дж.** Программирование для .NET. — М.: Русская Редакция, 2003.
10. **Уоткинз Д., Хаммонд М., Эйбрамз Б.** Программирование на платформе .NET. — М.: Вильямс, 2003.
11. **Gutknecht J., Zueff E.** Zonnon Language Experiment, or How to Implement a Non-Conventional Object Model for .NET // OOPSLA'02. — Seattle, Washington, 2002.
12. **Gutknecht J., Zueff E.** Zonnon Language Report. — Zurich, Institute of Computer Systems ETH Zentrum, 2004.
13. **Wirth N.** From Modula to Oberon // Software — Practice and Experience. — 1988. — Vol. 18, N 6. — P. 661 — 670.
14. **Wirth N.** The programming language Oberon // Software — Practice and Experience. — 1988. — Vol. 18, N 6. — P. 671 — 690.
15. **Zonnon** Builder. — Дистрибутив системы доступен по адресу: `http://www.zonnon.ethz.ch.`

ПРИЛОЖЕНИЕ

Ниже при описании синтаксиса языка Zonnon используется Расширенный Бекуса—Наура Формализм (РБНФ), который характеризуется следующими свойствами.

- Альтернативы разделяются символом |.
- Скобки [и] обозначают факультативность выражения в скобках.
- Скобки { и } обозначают повторение содержимого (возможно 0

раз).

- Скобки (и) используются для формирования групп элементов.
- Нетерминальные символы начинаются с прописной буквы (например, *Statement*).
- Терминальные символы либо начинаются со строчной буквы (например, *letter)*, либо записывается целиком полужирным шрифтом (например, **begin**), или представляются в виде строк (например, ":=").
- Комментарии начинаются с символа // и продолжаются до конца строки.

```
// Синтаксис Zonnon в РБНФ
// Версия от 11 марта 2004

// 1. Программа и программные единицы (Program and program units)
CompilationUnit = { ProgramUnit "." }.
ProgramUnit = ( Module | Definition | Implementation | Object ).

// 2. Модули (Modules)
Module = module [ ModuleModifier] ModuleName [ ImplementationClause ] ";"
    [ ImportDeclaration ]
    ModuleDeclarations
    ( BlockStatement | end ) SimpleName.
ModuleModifier = "{" ident "}" // private или public; private по умолчанию
ModuleDeclarations = { SimpleDeclaration | NestedUnit ";" }
    { ProcedureDeclaration | OperatorDeclaration }
    { ActivityDeclaration }.
NestedUnit = ( Definition | Implementation | Object ).
ImplementationClause = implements DefinitionName { "," DefinitionName }.
ImportDeclaration = import Import { "," Import } ";".
Import = ImportedName [ as ident ].
ImportedName = ( ModuleName | DefinitionName | ImplementationName |
    NamespaceName |ObjectName).

// 3. Определения (Definitions)
Definition = definition [ DefinitionModifier] DefinitionName [ RefinementClause ] ";"
    [ ImportDeclaration ]
    DefinitionDeclarations
    end SimpleName.
DefinitionModifier = "{" ident "}" // private или public; public по умолчанию
RefinementClause = refines DefinitionName.
DefinitionDeclarations = { SimpleDeclaration } { { ProcedureHeading ";" }|
    ActivitySpecification }.
ActivitySpecification =
    activity "{" ProtocolEBNF "}" ActivityName "=" EnumType ";".
```

```
ProtocolEBNF = Specification of the protocol in EBNF based on the syntax
    alphabet.

// 4. Реализации (Implementations)
Implementation = implementation [ImplementationModifier]
    ImplementationName ";" [ ImportDeclaration ]
    Declarations
    ( BlockStatement | end ) SimpleName.
ImplementationModifier = "{" ident "}". // private или public; public по умолчанию
// 5. Объекты
Object = object [ ObjModifier ] ObjectName [ FormalParameters ]
    [ ImplementationClause ] ";" [ ImportDeclaration ]
    Declarations
    { ActivityDeclaration }
    ( BlockStatement | end ) SimpleName.
ObjModifier = "{" ident "}". // value или ref; value по умолчанию
    // private или public; private по умолчанию
ActivityDeclaration = activity ActivityName [ ImplementationClause ] ";"
    Declarations
    ( BlockStatement | end SimpleName ).

// 6. Описания (Declarations)
Declarations = { SimpleDeclaration } { ProcedureDeclaration }.
SimpleDeclaration = ( const [DeclModifier] { ConstantDeclaration ";" }
    | type [DeclModifier] { TypeDeclaration ";" }
    | var [DeclModifier] { VariableDeclaration ";" }
    ).
DeclModifier = "{" ident "}". // public или private или immutable
ConstantDeclaration = ident "=" ConstExpression.
ConstExpression = Expression.
TypeDeclaration = ident "=" Type.
VariableDeclaration = IdentList ":" Type.

// 7. Типы (Types)
Type = ( TypeName [ Width ] | EnumType | ArrayType | ProcedureType |
    InterfaceType ).
Width = "{" ConstExpression "}".
ArrayType = array Length { "," Length } of Type.
Length = ( ConstExpression | "*" ).
EnumType = "(" IdentList ")".
ProcedureType = procedure [ ProcedureTypeFormals ].
ProcedureTypeFormals = "(" [ PTFSection { ";" PTFSection } ] ")"
    [ ":" FormalType ].
PTFSection = [ var ] FormalType { "," FormalType }.
FormalType = { array "*" of } ( TypeName | InterfaceType ).
InterfaceType = object [ PostulatedInterface ].
PostulatedInterface = "{" DefinitionName { "," DefinitionName } "}".
```

```
// 8. Процедуры и знаки операций (Procedures & operators)
ProcedureDeclaration = ProcedureHeading [ ImplementationClause ] ";"
    [ ProcedureBody ";" ].
ProcedureHeading = procedure [ ProcModifiers ]
    ProcedureName [ FormalParameters ].
ProcModifiers = "{" ident { "," ident } "}". // private, public, sealed
ProcedureBody = Declarations BlockStatement SimpleName.
FormalParameters = "(" [ FPSection { ";" FPSection } ] ")" [ ":" FormalType ].
FPSection = [ var ] ident { "," ident } ":" FormalType.
OperatorDeclaration = operator [ OpModifiers ] OpSymbol
    [ FormalParameters ] ";" OperatorBody ";".
OperatorBody = Declarations BlockStatement OpSymbol.
OpModifiers = "{" ident { "," ident } [ "," Priority ] "}" | "{" Priority "}".
Priority = ConstExpression.
OpSymbol = string. // 1,2-литерная строка;
    // множество возможных символов ограничено

// 9. Операторы (Statements)
StatementSequence = Statement { ";" Statement }.
Statement = [ Assignment
    | ProcedureCall
    | IfStatement
    | CaseStatement
    | WhileStatement
    | RepeatStatement
    | LoopStatement
    | ForStatement
    | await Expression
    | exit
    | return [ Expression ]
    | BlockStatement
    | launch Statement
    | Send
    | BlockingReceive
    | NonBlockingReceive
    ].
Assignment = Designator ":=" Expression.
ProcedureCall = Designator.
IfStatement = if Expression then StatementSequence
    { elsif Expression then StatementSequence }
    [ else StatementSequence ]
    end.
CaseStatement = case Expression of
    Case { "|" Case }
    [ else StatementSequence ]
    end.
```

```
Case = [ CaseLabel { "," CaseLabel } ":" StatementSequence ].
CaseLabel = ConstExpression [ ".." ConstExpression ].
WhileStatement = while Expression do StatementSequence end.
RepeatStatement = repeat StatementSequence until Expression.
LoopStatement = loop StatementSequence end.
ForStatement = for ident ":=" Expression to Expression [ by ConstExpression ]
    do StatementSequence end.
BlockStatement = begin [ BlockModifiers ]
    StatementSequence
    { ExceptionHandler }
    [ CommonExceptionHandler ]
    end.
BlockModifiers = "{" ident { "," ident } "}". // locked, concurrent
ExceptionHandler = on ExceptionName { "," ExceptionName } do
    StatementSequence.
CommonExceptionHandler = on exception do StatementSequence.
Send = send expression [ "=>" activity].
BlockingReceive = receive [ activity "=>" ] variable.
NonBlockingReceive = accept [ activity "=>" ] variable.

// 10. Выражения (Expressions)
Expression = SimpleExpression
    [ ( "=" | "#" | "<" | "<=" | ">" | ">=" | in ) SimpleExpression ]
    | Designator implements DefinitionName
    | Designator is TypeName.
SimpleExpression = [ "+"|"-" ] Term { ( "+" | "-" | or ) Term }.
Term = Factor { ( "*" | "/" | div | mod | "&" ) Factor }.
Factor = number
    | CharConstant
    | string
    | nil
    | Set
    | Designator
    | new TypeName [ "(" ActualParameters ")" ]
    | new ActivityInstanceName
    | "(" Expression ")"
    | "~" Factor.
Set = "{" [ SetElement { "," SetElement } ] "}".
SetElement = Expression [ ".." Expression ].
Designator = Instance
    | Designator "{" Type "}" // Casting
    | Designator "^" // Dereference
    | Designator "[" Expression { "," Expression } "]" // Array element
    | Designator "(" [ ActualParameters ] ")" // Function call
    | Designator "." MemberName // Member selector
Instance = ( self | InstanceName | DefinitionName "(" InstanceName ")" ).
ActualParameters = Actual { "," Actual }.
```

```
Actual = Expression [ "{" [ var ] FormalType "}" ].
    // Аргумент с сигнатурой типа

// 11. Constants
number = (whole | real) [ "{" Width "}" ].
whole = digit {digit} | digit {hexDigit} "H".
real = digit { digit } "." { digit } [ ScaleFactor ].
ScaleFactor = "E" ["+" | "" ] digit { digit }.
HexDigit = digit | "A" | "B" | "C" | "D" | "E" | "F".
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
CharConstant = "'" character "'" | "'" character "'" | digit { HexDigit } "X".
string = '"' { character } '"' | "'" { character } "'".
character = letter | digit | Other.
Other = // Любая литера из алфавита за исключением тех,
    // которые используются…

// 12. Идентификаторы и имена (Identifiers & names)
ident = ( letter | "_" ) { letter | digit | "_" }.
letter = "A" | ... | "Z" | "a" | ... | "z" |
    // любая другая "культурно-определенная" буква
IdentList = ident { "," ident }.
QualIdent = { ident "." } ident.
DefinitionName = QualIdent.
ModuleName = QualIdent.
NamespaceName = QualIdent.
ImplementationName = QualIdent.
ObjectName = QualIdent.
TypeName = QualIdent.
ExceptionName = QualIdent.
InstanceName = QualIdent.
ActivityInstanceName = QualIdent.
ProcedureName = ident.
ActivityName = ident.
MemberName = ( ident | OpSymbol ).
SimpleName = ident.
```